intel

APPLICATION NOTE

AP-61

July 1979

Multitasking for the 8086

Cecil Moore
Applications Engineer
Microprocessor Products

Multitasking For the 8086

Contents

INTRODUCTION

Real-time software systems differ markedly from batch processing systems. An external signal indicating that it is time for an hourly log or an interrupt caused by an emergency condition is an event usually not encountered in batch processing. Because real-time control systems of all types share a number of characteristics, it is possible to develop flexible operating systems which will meet the needs of a great majority of real-time applications. Intel Corporation has developed such a system, the RMX/80™ system, for the iSBC™ line of 8080/85 based single board computers. Thus, the user is released from the chore of designing an operating system and is free to concentrate his efforts on the applications software for the individual tasks and merely integrate them into a pre-existing system.

But what if a user does not need all the capabilities of an RMX/80™ system or wants a different hardware configuration than an iSBC™ computer? This application note contains a set of PL/M-86 procedures designed to be used in medium-complexity 8086 real-time systems.

A normal control system can be broken down into a number of concurrently executable tasks. The CPU can be running only one task at any instant of time but the speed of the processor often makes concurrent tasks appear to be running simultaneously. Breaking the software functions into separate concurrent tasks is the job of the designer/programmer. Once this is done there remains the problem of integrating these tasks with a supervisory program which acts as a traffic cop in the scheduling and execution of the separate tasks. This note discusses a set of PL/M-86 procedures to implement the supervisory program function.

A minimum operating system might (like its batch processing cousin) have only a queue for ready tasks (tasks waiting to be executed). Any task that becomes ready is put on the bottom of the queue and when a running task is finished, the task on the top of the queue is started. Any interrupt causes the state of the system to be saved, an interrupt routine to be executed, the state of the system to be restored, and execution of the interrupted program to continue. The interrupt routine might (or might not) put a new task on the ready queue. This approach has worked well for many simple control systems, especially in the single-chip computer area. But what features are lacking in this approach that are necessary (or at least nice)?

1. A system of priorities is often needed. All waiting ready tasks must be executed sooner or later but some tasks need immediate attention while others can be run when there is nothing else to do. If a midnight monthly report, due for completion by 8 a.m. the next day, is in the process of printing at 1 a.m. and a fire alarm occurs, it is reasonable to assume that the fire alarm has higher priority since the fire could conceivably render the monthly report irrelevant.

There are a number of ways in which to assign priorities. Tasks are usually numbered and may be assigned priorities according to their ascending (or descending) numbers. They could instead be grouped into a number of priority levels, with tasks on the same level having equal priorities. The latter approach is taken in this application note.

Assume that a monthly report is being printed and an alarm occurs in the external world that, because of its importance, must be attended to immediately. The interrupt routine, executed as a result of the alarm input, should not automatically return to the interrupted logging routine but instead should call a preempt routine which checks to see if a higher priority task is ready for execution. The reason for this is that the monthly report routine, if returned to, has no way of "knowing" that a higher priority task is waiting to be executed. The alarm output task has been readied by the interrupt routine and since it is known to be higher priority than the logging task, it is executed first, thereby immediately signaling the system operator that there has been an alarm. It then returns to the logging task provided that there are no further high priority tasks waiting to be executed. The logging printer may not have even paused during the alarm output task. The computer appears to human beings to be executing concurrent tasks simultaneously.

Of course, the alarm output function could be performed inside the interrupt procedure. But sooner or later, the designer will encounter a worst case situation in which there is not enough time to execute all required tasks between interrupts, and the system will fall behind in real-time. It is much cleaner to make the interrupt procedures as short as possible and stack up tasks to be executed than to stack up interrupt procedures.

2. Another feature that might be necessary is a capability to put a task to sleep for a known period of real time. Assume a relay output must remain closed for one second. Most real-time systems cannot tolerate the dedication of the CPU to such a trivial task for that length of time so a system of programmable dynamic delays could be implemented. This application note implements such a system.

Although the PL/M-86 procedures here have been debugged and tested, it is assumed that the user will want to change, add, or delete features as needed. This application note is intended to present ideas for a logical structure of procedures that, because they are written in PL/M-86, can be easily modified to user requirements. Each procedure will be discussed in detail and integration and optional features will be presented.

PL/M-86

PLM-86 is a block structured high level language that allows direct design of software modules. Using PL/M-86, designers can forget their assembly level

coding problems and design directly in a subset of the English language. The 8086 architecture was designed to accommodate highly structured languages and the PLM-86 compiler is quite efficient in the generation of machine code.

PLM-86 STRUCTURE

PL/M-86 automatically keeps track of the level of the different software blocks. (See Chapter 10, "PL/M-86 Programming Manual"). There are methods of writing PL/M-86 which contribute to the understandability of the source code without adding to the amount of object code generated. For instance, the following three IF/THEN/ELSE blocks generate identical object code but are compiled from different source statements.

Line	Level	Statement
3	1	IF A = B THEN C = D; ELSE E = F; G = H;
7	1	IF A = B THEN
8	1	C = D;
		ELSE
9	1	E = F;
10	1	G = H;
11	1	IF A = B THEN DO;
13	2	C = D ;
14	2	END;
15	1	ELSE DO;
16	2	E = F;
17	2	END;
18	1	G = H;

It is not instantly apparent from the code on line 3 or the code starting at line 7 which statements will be executed. However, adding the DO; and END; statements (starting at line 11) remove any doubt. Either the statements starting at line 11 or the statements starting at line 15 will be executed and the statement on line 18 will be executed in either case. Why? Because all these lines are at level 1 in the block structure. The other lines are at level 2 because of the DO;/END; combinations. When one refers to the relatively complex structures of the task multiplexer procedures, the usefulness of such an approach is obvious, as the procedures have been indented according to the level numbers generated by PL/M-86. In particular, if the designer is not careful, nested IF/THEN/ELSE statements can generate improper results. Using a proper number of DO;/END; combinations avoids the possible ambiguity in nested IF/THEN/ELSE statements as can be seen in the ACTIVATE$TASK procedure listed in the PL/M-86 source code later in this note. The DO;/END; construct naturally must be used when multiple statements are required within the IF/THEN/ELSE blocks. Following are examples of the possible primary structures of PL/M-86:

```
DO;
    A = B;
    C = D;
    END;

DO WHILE A = B;
    C = D;
    E = F;
    END;

DO I = 1 TO 5;
    A = I;
    C = D + I;
    END;

DO CASE A;
    A = B;
    A = C;
    A = D;
    END;

IF  A = B THEN DO;
    C = D;
    END;
ELSE DO;
    E = F;
    END;

IF  A = B THEN DO;
    C = D;
    END;
ELSE IF A = C THEN DO;
    D = E;
    END;
ELSE IF A = D THEN DO;
    E = F;
    END;
ELSE DO;
    F = G;
    END;
```

A complete tutorial on structured programming is beyond the scope and intent of this application note and the reader is referred to the appropriate references appearing in the bibliography.

ANATOMY OF THE TASK MULTIPLEXER

Once a decision is made on the details of the kind of data structure that is needed to implement the task multiplexer, the procedures that manipulate the structure are relatively simple to write. The following characteristics are assumed for the task multiplexer appearing in this application note.

There are two levels of priority, high and low. All high priority tasks that are ready to run will be dispatched, executed, and completed, on a FIFO basis, before any low priority task is dispatched.

Any task can be interrupted. No task multiplexer procedure can be interrupted.

If a high priority task is interrupted, it will be completed before any other task is dispatched. If a low priority task is interrupted, all ready high priority tasks will be dispatched, executed, and completed before program control is returned to the low priority task.

There are two ready queues, one for high priority tasks and one for low priority tasks. Each queue has a head (top) pointer and a tail (bottom) pointer and tasks on any queue are link-listed from head to tail. Tasks are "dispatched" (taken off the queue) at the head and "activated" (put on the queue) at the tail on a FIFO basis.

Link-listed queues are chosen for simplicity. All dispatch and activate information is contained in the head and tail pointers. Tasks located in the middle of these link-lists are of no concern for activating and dispatching. This means, of course, that tasks are executed in the order that they appear on the queue, i.e., first-in, first-out.

There is a pointer byte associated with each task. If a task is on either the low priority or high priority ready queue, its associated pointer byte will point to the next task number on the list. These pointer bytes enable the task ready lists to be linked. Note that the pointer byte is 0 for the last task on a list.

There is a status (flag) byte associated with each task. If a task is on a ready list or a delay list, bit 7 will be a "1" indicating that that particular task is busy. If a task is on either high priority or low priority ready queues, bit 6 will be a "1" indicating that the task is on one of the ready queues. If the task is listed on the delay list, (see next item), bit 5 will be a "1" indicating that this particular task has a delay in progress. If a task is unlisted, bits 5–7 will be "0." Bits 0–4 are not used by the task multiplexer procedures and are available to the user, giving 5 user defined flags per task.

There is a delay byte associated with each task. This feature allows tasks to be "put to sleep" for a variable length of time, from 1 to 255 "ticks" of the interrupt clock. If a task does not need an associated delay then this byte is available to the user as a utility byte to be used for any purpose. These delays will be discussed in detail later in the application note.

The following diagram is a representation of the task multiplexer data structure:

TASK NUMBER	POINTER BYTE	STATUS BYTE	DELAY BYTE
0	n	n + 1	n + 2
1	n + 3	n + 4	n + 5
2	n + 6	n + 7	n + 8
3	n + 9	n + 10	n + 11
4	n + 12	n + 13	n + 14
5	n + 15	n + 16	n + 17
m – 1	n + 3m – 6	n + 3m – 5	n + 3m – 4
m	n + 3m – 2	n + 3m – 1	n + 3m

3m + 3 TOTAL RAM BYTES
n = FIRST RAM ADDRESS OF ARRAY

Following is a chart of what a task multiplexer data structure might look like at a given moment in time:

HIGH$PRIORITY$HEAD = 5
HIGH$PRIORITY$TAIL = 3
LOW$PRIORITY$HEAD = 8
LOW$PRIORITY$TAIL = 10
DELAY$HEAD = 4

TASK NUMBER	TASK(n).PNTR	TASK(n).STATUS	TASK(n).DELAY
0	*	*	*
1	3	1100 0000	0
2	0	1010 0000	3
3	0	1100 0000	0
4	7	1010 0000	4
5	1	1100 0000	0
6	0	0000 0000	0
7	2	1010 0000	6
8	10	1100 0000	0
9	0	0000 0000	0
10	0	1100 0000	0

*See text.

What information can one ascertain from observation of the above chart? The ready-to-run high priority tasks, in order, are 5,1,3. This can be seen by following the high priority ready linked list from head to tail. The ready-to-run low priority tasks, in order are 8, 10. The TASK(n).PNTR byte = 0 for the last listed task. Tasks 4, 7, 2 are listed, in order, on the delay list and have associated delays of 4, 10, 13 ticks respectively. Tasks 6 and 9 are not listed and therefore idle. The * for the TASK (0) bytes indicate a special condition. There is no TASK00 allowed and a zero condition is treated as an error condition. TASK(0).PNTR byte is used for the DELAY$HEAD byte to minimize code in the ACTIVATE$DELAY procedure. TASK(0).STATUS and TASK(0).DELAY are unused bytes.

DEFINITIONS

NEW$TASK is the number of the task that will be installed on a ready list or the delay list when ACTIVATE$TASK or ACTIVATE$DELAY is called.

NEW$DELAY is the value of the delay that will be installed on the delay list when ACTIVATE$DELAY is called.

A task is defined as RUNNING if it is in the act of execution or if an interrupt routine is executing which interrupted a RUNNING task.

A task is defined as PREEMPTED if it has been interrupted and a higher priority task is being executed.

A task is defined as READY if it is contained within one of the ready queues.

A task is defined as IDLE if its BUSY$BIT (bit 7) is not set, i.e., it is not listed anywhere else. Note that it is possible to completely disable an IDLE task simply by setting its BUSY$BIT. In that case, it is not and cannot be listed anywhere else. This feature is useful during system integration.

STATE DIAGRAM

The state diagram indicates the relationships among the possible task states and the procedures involved in changing states.

The state diagram looks somewhat complicated and a discussion of the possible change of states is in order. Assuming a certain existing state, future possible states will be discussed including the procedures which can cause the change of state.

From the unlisted (idle) state, the ACTIVATE$TASK procedure will put the NEW$TASK on either the high priority ready queue or the low priority ready queue at the tail end of the queue. The number of the task automatically assigns the priority and therefore the proper queue. All task numbers below FIRSTLOWPRIORITY$TASK are assumed to be high priority tasks. Also, from the unlisted state the ACTIVATE$DELAY procedure will put the NEW$TASK and NEW$DELAY at the proper position on the delay list.

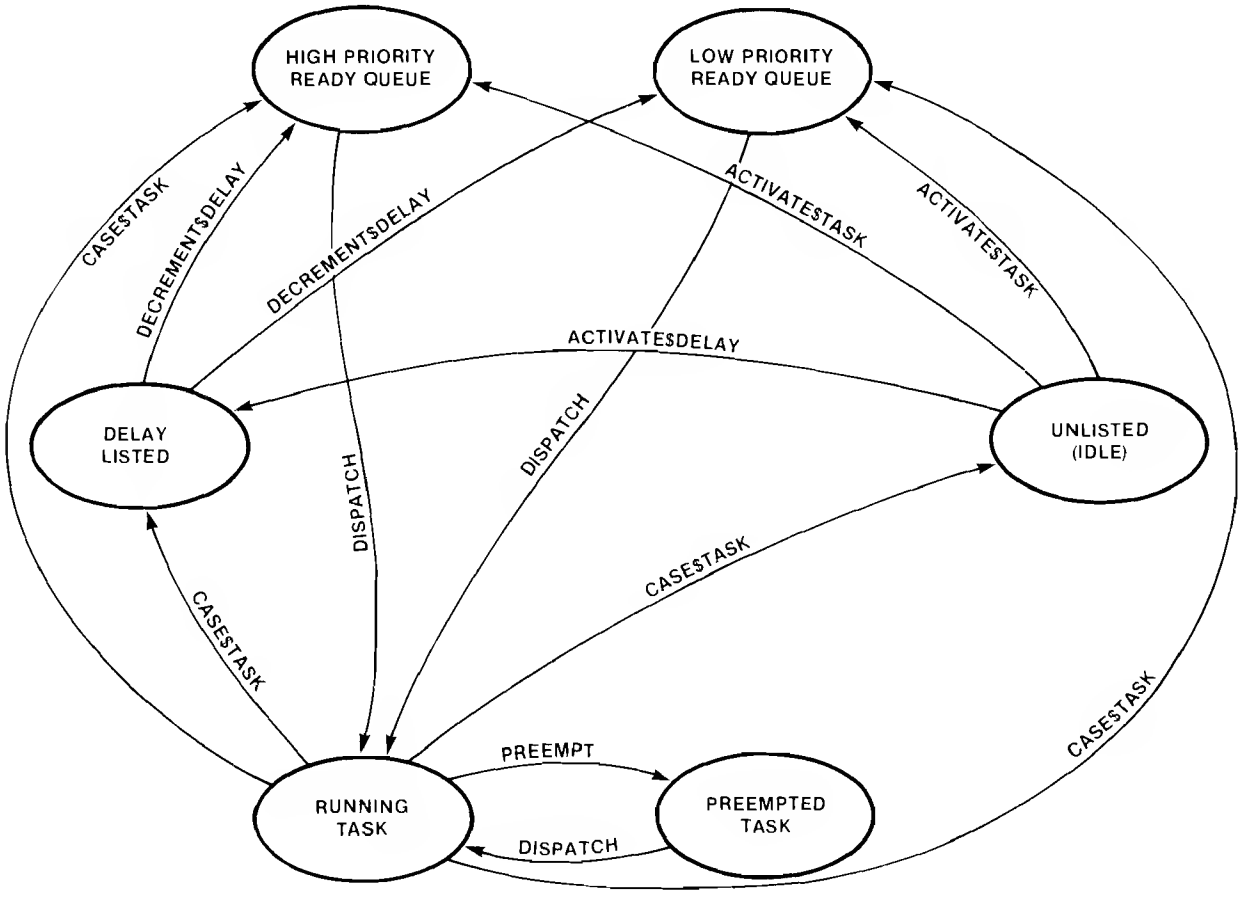
After a task has been put on either high priority ready queue or low priority ready queue it eventually will go to the RUNNING$TASK state. The DISPATCH procedure accomplishes this action.

From the delay list a task can only go to one of the ready queues. When a task's associated delay goes to zero the DECREMENT$DELAY procedure calls the ACTIVATE$TASK procedure and installs the NEW$TASK on the proper ready queue.

From the RUNNING$TASK state a task may use the CASE$TASK procedure to put itself on the ready list tail by setting NEW$TASK = RUNNING$TASK. It may instead put itself on the delay list by setting NEW$TASK = RUNNING$TASK and also setting NEW$DELAY equal to something other than zero. Otherwise, it will progress to the unlisted state upon completion.

The CASE$TASK procedure unlists tasks when they have completed execution. A low priority RUNNING$TASK will go to the preempted state if a high priority task is on the ready list following an interrupt during execution of the low priority task if the PREEMPT procedure is called.

And finally, a PREEMPTED$TASK will return to a RUNNING$TASK state when all high priority ready tasks have completed execution. This is accomplished by the DISPATCH procedure which then returns to the PREEMPT procedure.



STATE DIAGRAM

Some lockouts are necessary to avoid chaos in the task multiplexer. These are as follows:

The BUSY$BIT = 1 in the TASK(n).STATUS byte will abort the ACTIVATE$TASK and the ACTIVATE$DELAY procedures and return an indication of the aborting by setting the STATUS byte equal zero. A task must be unlisted to be able to be installed on a list.

A RUNNING$TASK may put itself on a list after it has executed but it is not allowed to re-list any listed tasks (i.e., no task may ever be listed twice at the same time!). A task that tries to activate another task that is already busy can wait (via the delay feature) for the required task to complete execution, become idle, and therefore be available to be activated. A PREEMPTED$TASK may not be listed. If the ACTIVATE$TASK or ACTIVATE$DELAY procedure is called and NEW$TASK = PREEMPTED$TASK, the procedure will be aborted and return with STATUS = 0. Otherwise, the STATUS byte is returned with the new task status.

Only one task may be preempted as there are only two levels of priority. The user may desire to implement many levels of priority in which case a linked-list of preempted tasks could be declared in a structure which includes the number of the first task in each priority level group of tasks. This obviously complicates the PREEMPT and DISPATCH procedures.

The tasks themselves are made into reentrant procedures because of the necessary forward references of the CASE$TASK procedure.

PL/M-86 allows structures and arrays of structures. The structure needed for the task multiplexer is a link-list pointer byte, a task status byte, and a task delay byte. Each task has an associated pointer byte, status byte, and delay byte. These are combined into an array of up to 255 tasks. For purposes of this discussion, the number of tasks is chosen as an arbitrary 10, leading to the following array declaration.

```
DECLARE TASK(10)STRUCTURE
(PNTR BYTE,STATUS BYTE,DELAY BYTE);
```

Thus the delay byte associated with task number 7 can be accessed by using the variable TASK(7).DELAY and the status of task number 5 can be examined through the use of TASK(5).STATUS. The TASK(n).PNTR byte contains the task number of the next listed task on the same list as TASK(n), i.e., if TASK(n) is on the delay list, then TASK(n).PNTR will contain the number of the next task on the delay list or 0 indicating the end of the list.

TASK(n).STATUS is a byte with the following reserved flags:

```
BIT 7   BUSY$BIT, "1" IF TASK IS BUSY
BIT 6   READY$BIT, "1" IF ON READY LIST
BIT 5   DELAY$BIT, "1" IF ON DELAY LIST
BIT 4 — BIT 0     UNUSED
```

The unused bits in the STATUS byte are available to the user.

The TASK(n).DELAY byte is a number which can put TASK(n) to sleep for up to 255 system clock ticks. The system clock tick is interrupt driven from the user's timer and its period is chosen for the particular application. A one millisecond timer is popular and assuming such a time, delays of up to 255 ms are available in the task multiplexer as it is written. If this delay range is not wide enough, the user may want to define his TASK(n).DELAY as a word instead of a byte in the PL/M-86 declare statement, giving delays of up to 65 seconds from the basic one millisecond clock tick.

LINKED LISTS

Linked lists are useful for a number of reasons. However, a treatise on linked lists would defeat the purpose of this application note and the reader is referred to the references listed in the bibliography.

The linked lists used in this application note have a head byte associated with each list, i.e., the head byte contains the number of the first task on the list. The first task pointer byte points to the second task on the list, etc. The pointer of the last task on the list is set at zero to indicate that it is the last task. Two of the linked lists are ready queues and require a tail byte as well as a head byte. The tail byte points to the last entry on the list. Tasks are put on the bottom, or tail, of the ready lists and are taken off the top, or head, of the ready lists. The delay list has no tail but does have a head, called a DELAY$HEAD. The delay list is not a queue, as delays are installed on the list in order of delay magnitude for reasons to be explained later.

There are two ready lists, one for high priority tasks and one for low priority tasks. The head and tail pointers associated with these two lists are: HIGH$PRIORITY$HEAD, HIGH$PRIORITY$TAIL, LOW$PRIORITY$HEAD, and LOW$PRIORITY$TAIL. Obviously, the structure can be expanded to any number of priority levels by expanding the head and tail pointers and the historical record of the preempted tasks.

DELAY STRUCTURE

A task multiplexer can have a number of simultaneous delays active and it would be efficient if there were a way to keep from decrementing all delays on every clock tick, which is most time consuming. One way to accomplish this feat is to move the problem from the DECREMENT$DELAY routine to the ACTIVATE$DELAY routine. The delays are arranged in a linked-list of ascending sizes such that the value of each delay includes the sum of all previous delays. This allows the decrementing of only one delay during each clock tick interrupt routine. An example will further illuminate this approach. Suppose the following conditions exist:

Task 7 has a 5 millisecond delay

Task 3 has an 8 millisecond delay

Task 9 has a 14 millisecond delay

The delay structure is arranged so that:

DELAY$HEAD = 07
TASK(7).PNTR = 03
TASK(3).PNTR = 09
TASK(9).PNTR = 00
TASK(7).DELAY = 05 (FIRST DELAY = 5)
TASK(3).DELAY = 03 (5 + 3 = 8)
TASK(9).DELAY = 06 (5 + 3 + 6 = 14)

The linked-list is arranged so that the delays are in ascending order and each delay is equal to the sum of all previous delays up through that point. Since this is true, all delays are effectively decremented merely by decrementing the first delay. Of course, something for nothing is impossible and the speed gained by arranging the delays in the above manner is paid for by the complexity of the ACTIVATE$DELAY routine. But since the ACTIVATE$DELAY routine is executed less frequently than the DECREMENT$DELAY routine, the savings in real time is worth the added complexity.

Suppose a new delay is to be activated in the above scheme. Task 5 with a delay of 10 milliseconds is to be acded. A before and after chart will indicate what the ACTIVATE$DELAY procedure must accomplish.

BEFORE

TASK NUMBER		07	03	09	
POINTER	07	03	09	00	
DELAY		05	03	06	

AFTER

TASK NUMBER		07	03	05	09
POINTER	07	03	05*	09@	00
DELAY		05	03	02@	04*

FIRST POINTER IS THE DELAY$HEAD
CHANGES ARE MARKED WITH AN *
ADDITIONS ARE MARKED WITH AN @

Note that the pointer before the added task has changed and the delay after the added task has changed. The function of the ACTIVATE$DELAY procedure is to accomplish these changes and additions.

PROCEDURES

The following procedure explanations reference the PL/M-86 source code listing which follows the application note text.

ACTIVATE$TASK Procedure

This procedure is initiated by a call instruction with the byte NEW$TASK containing the number of the task to be put on the proper ready queue.

Interrupts must be disabled whenever the link-lists are being changed. If interrupts are enabled when this procedure is called, they should be re-enabled upon returning.

The assignment of priority is a simple matter. A declare statement, DECLARE FIRSTLOWPRIORITY$TASK LITERALLY 'N,' (where N is the actual number of the first low priority task) indicates to the procedures that tasks 1 to N are high priority tasks and tasks N or higher are low priority tasks.

This procedure checks the busy bit in the status byte to see if this particular task is already busy and if so, returns a STATUS of zero. Otherwise, it returns the new STATUS of the task. It then checks the priority to see if this particular task is a high or low priority. If it is high priority, then the task pointer pointed to by the HIGH$ PRIORITY$TAIL pointer is changed from zero to the number of the NEW$TASK. The HIGH$PRIORITY$TAIL pointer is then changed to the number of the NEW$TASK and the pointer associated with NEW$ TASK is made equal to zero. This completes the ACTIVATE$TASK functions. If the new task is a low priority task, then the same functions are performed using the LOW$PRIORITY$TAIL pointer.

ACTIVATE$DELAY Procedure

This procedure is initiated by a call with the byte NEW$ TASK containing the number of the task to be put on the delay list and the byte NEW$DELAY containing the value of the associated delay.

Interrupts are disabled and the busy bit of this particular task is checked. If the busy bit is set the STATUS byte is set to zero and the procedure returns without activating the delay. If the busy bit is not set the integer value DIFFERENCE is set equal to the NEW$DELAY value. POINTER$0 is set equal to the DELAY$HEAD. POINTER$1 is set to zero. The DO WHILE loop executes until POINTER$0 equals zero or DIFFERENCE is less than zero. Remember that the proper place to insert the new delay is being searched for, and that will be either at the end of the list (POINTER$0 = 0) or when the sum of the previous delays do not exceed the new delay value. The DO WHILE loop has POINTER$0, POINTER$1, OLD$DIFFERENCE, and DIFFERENCE keeping track of where the procedure is in the loop, while searching for the proper place to insert the new delay. The existing delays are sequentially subtracted from the remains of NEW$ DELAY according to the link-listed order until the end of the list or a negative result is encountered indicating that the proper delay insertion point has been reached. At this point POINTER$0 contains the task number to be assigned to TASK(NEW$TASK).PNTR. POINTER$1 contains the task number immediately preceding the NEW$TASK such that TASK(POINTER$1). PNTR = NEW$ TASK and our link list is fully updated, with the actual delays yet to go. If POINTER$0 = 0 it means that the new delay is larger than any of the other delays and therefore should go on the end of the list so TASK(NEW$ TASK).DELAY is set equal to the DIFFERENCE. If

POINTER$0 is not equal to zero then if POINTER$0 equals POINTER$1 (indicating that there were not any delays previously listed), then TASK(POINTER$1).PNTR is set equal to zero. TASK(NEW$TASK).DELAY is set equal to the OLD$DIFFERENCE and TASK (POINTER$0).DELAY is set equal to the negative of DIFFERENCE which at this point is negative, thereby resulting in a positive unsigned number. The reader is encouraged to implement an example (see Delay Structure section) to prove that the above approach is valid. Particular attention should be paid to the contents of the two pointers, as they are the key to the procedure. The final function of this procedure is to set the BUSY$BIT and DELAY$BIT in the TASK(NEW$ TASK).STATUS byte. The byte named STATUS which is returned by this procedure is set equal to the status of the new task. If it is desired to have interrupts enabled, they must be enabled after the procedure return instruction. The reason for such a complex method of activating a delay will become apparent in the following section.

DECREMENT$DELAY Procedure

The first delay on the linked-list is decremented and, if it is zero, the associated task is put on the appropriate ready queue. The next delay (if any) is checked to see if it is zero and if so, that task is put on the appropriate ready queue, etc. A loop is performed until either no delay or a non-zero delay is found. The procedure then returns.

It is assumed that this procedure is part of an interrupt routine and that the interrupts are disabled during its execution. Interrupts cannot be enabled during changes to any of the linked-lists or else recovery may not be possible.

This procedure begins by checking to see if there are any active delays. If DELAY$HEAD = 0 then this procedure returns immediately. Otherwise it decrements the first delay. If this delay goes to zero then the associated task number is passed to the ACTIVATE$ TASK procedure as the OFF$DELAY byte. A new DELAY$HEAD is chosen from the next link-listed delay and that delay checked for a value of zero which will happen if the first two or more delays are equal. This loop is accomplished by the DO WHILE DELAY$ HEAD <> 0 AND TASK(DELAY$HEAD).DELAY = 0; This procedure is designed to require very little CPU time unless a delay times out. The DO WHILE loop is bypassed if the resulting delay value is not zero. A certain amount of care should be exercised to insure that many delays do not all time out at the same time. One method would be to modify the ACTIVATE$DELAY procedure to insure that there are no zero entries in the delay bytes. The basic procedure, however, assumes that the clock "tick" timing will be chosen to minimize the above potential problem.

CASE$TASK Procedure

This procedure performs the function of calling the task indicated by the contents of the RUNNING$TASK byte. All listed tasks are called in this manner. The CASE$TASK procedure is called by the DISPATCH procedure. When a particular task has completed execution it returns to the CASE$TASK procedure which then resets the BUSY$BIT and the READY$BIT and returns to the DISPATCH procedure after setting RUNNING$TASK equal to zero. This procedure allows a task to relist itself immediately upon returning from execution.

PREEMPT PROCEDURE

The PREEMPT procedure is called whenever it is possible that a high priority task has been put on the ready queue while a low priority task was in the process of execution. An example will illustrate:

Assume that the control system is being interrupted by the 60 Hz line frequency and a register is being incremented each time this 16.67 ms edge occurs. When the register gets to 60 (indicating that one second has passed), the register is zeroed and the high priority timekeeping task is put on the ready queue. Assume also that a low priority data logging task was running when this interrupt occurred. The interrupt routine calls PREEMPT. If a high priority task is running, PREEMPT simply returns. But in our example, a low priority task is running so PREEMPT transfers RUNNING$TASK to PREEMPTED$TASK and calls DISPATCH, which calls CASE$TASK, which calls the time-keeping task. When the time-keeping task has completed, it returns to CASE$TASK which returns to DISPATCH which returns to the PREEMPT procedure which returns to the interrupt routine which returns to the interrupted low priority data logging task if no other high priority tasks are on the ready queue. If the high priority ready queue is not empty, any and all high priority tasks will be completed before the interrupted routine is returned to. PREEMPT refuses to return to the interrupt routine until HIGH$ PRIORITY$HEAD is equal to zero. It is important to note that a low priority task will not be preempted unless the PREEMPT procedure is called. As noted above, it is normally called from the interrupt routine which interrupted the low priority task, but there is nothing to prohibit PREEMPT from being called from inside a low priority task procedure.

DISPATCH PROCEDURE

This procedure calls a high priority task if HIGH$ PRIORITY$HEAD is not equal to zero, restores a preempted task if PREEMPTED$TASK is not equal to zero, calls a low priority task if LOW$PRIORITY$HEAD is not equal to zero, and simply returns if there is nothing to do, all in order of priority. The DISPATCH procedure is called from the main program loop which must enable interrupts as DISPATCH disables interrupts as soon as

it is called. It is also called by the PREEMPT procedure. RUNNING$TASK must be 0 when this procedure is called.

PL/M-86 PROCEDURES

Because the block structure and levels are so important to the understanding of the following procedures, they have been indented according to level. This was a simple task accomplished by no indenting for level one, indenting once for level two, etc. The resulting attractive, easy to follow format was worth the effort to increase the initial level of understanding for readers of this application note who are not intimately familiar with PL/M.

Everything except the very simple main program loop has been made into procedures. Interrupt routines and tasks are also procedures. Keeping track of interrupts, calls, and returns is easy for PL/M and a violation of the block structure through such devices as GOTO targets outside the procedure body is the best way the author knows to crash and burn. Honor the power of the structure accept the limitations involved, and checkout and debugging will be a pleasure.

Since CASE$TASK references the individual tasks, the task procedure structure was included in the PL/M-86 compilation. All the user has to do is insert the particular task code in place of the /*TASKnn CODE*/ comment, define the interrupt procedures and the system should be ready to run. Obviously, the user will desire to change the total number of tasks and the number of the FIRSTLOWPRIORITY$TASK.

INITIALIZATION AND THE MAIN LOOP

The last entry in the PL/M-86 program is the initialization process which essentially zeros the task multiplexer data and the main loop which loops until TRUE = FALSE, i.e. forever, with interrupts enabled. The STATUS = STATUS instruction simply insures that the loop can be interrupted as the instruction following an ENABLE instruction is not interruptible.

These few instructions are included for information only and will need to be expanded considerably for use in a real-world system. The task multiplexer procedures were checked out on an iSBC 86/12™ computer running under random interrupt control and these instructions were the minimum necessary to cause the system to run. As was stated earlier, the following source code does not include any interrupt procedures and these will have to be generated following the format explained in the PL/M-86 programming manual.

ADDITIONAL IDEAS

Resource allocation is a feature that could be added to the task multiplexer. To keep it simple and yet avoid the deadlock problem (two tasks each grab a resource that the other needs), an extra array can be added to the TASK(n).XXX structure in which each bit in the byte (or word), represents a resource necessary for the execution of a task. A RESOURCES$STATUS byte can then keep the dynamic busy status of the system resources (printers, terminals, floating point math packages, etc.). When the CASE$TASK procedure is called, the resources required by the next RUNNING$TASK can be compared to the RESOURCES$STATUS byte to see if the required resources are available. If they are, the following PL/M-86 statement will update the new status of the resources:

```
RESOURCES$STATUS = RESOURCES$STATUS OR
        TASK(RUNNING$TASK).RESOURCES;
```

However, if the resources are not available, the CASE$TASK procedure can return the task to the ready or delay list and try again later. When the task has completed, the following PL/M-86 statement will update the resources status byte:

```
RESOURCES$STATUS = RESOURCES$STATUS AND NOT
        TASK(RUNNING$TASK).RESOURCES;
```

Message passing from task to task may also be necessary. Assuming that a task will have only one message at a time to deliver or receive, another byte could be added to the task structure such that TASK(RUNNING$TASK).MESSAGE could represent a byte containing the number of the task wishing to deliver a message to the RUNNING$TASK. Since a task can call CASE$TASK which in turn will call another task, message block parameters can be passed directly from one task to another. The task that calls CASE$TASK must handle the necessary housekeeping involved in recovering after the message has been passed. Of course, the data structure would have to be expanded to accommodate the message parameters and blocks. For further ideas involving message handling refer to the RMX/80™ user's guide.

Two additional relatively simple procedures could be added to obtain the SUSPEND and RESUME features of the RMX/80™ system. Remember that if the BUSY$BIT is set in a TASK(n).STATUS byte and the task is unlisted, then it cannot be listed. If it is desired to dynamically enable and disable a task, this bit could be set by a SUSPEND procedure and reset by the RESUME procedure.

```
TM86:DO;

DECLARE TOTAL$TASKS LITERALLY '10';
DECLARE TRUE LITERALLY '0FFH';
DECLARE FALSE LITERALLY '0';
DECLARE BUSY$BIT LITERALLY '10000000B';
DECLARE READY$BIT LITERALLY '01000000B';
DECLARE DELAY$BIT LITERALLY '00100000B';
DECLARE FIRST$LOW$PRIORITY$TASK LITERALLY '6';

DECLARE TASK(TOTAL$TASKS) STRUCTURE(PNTR BYTE, STATUS BYTE, DELAY BYTE);
DECLARE HIGH$PRIORITY$HEAD BYTE, HIGH$PRIORITY$TAIL BYTE;
DECLARE LOW$PRIORITY$HEAD BYTE, LOW$PRIORITY$TAIL BYTE;
DECLARE RUNNING$TASK BYTE, PREEMPTED$TASK BYTE;
DECLARE STATUS BYTE, NEW$TASK BYTE, NEW$DELAY BYTE;
DECLARE DELAY$HEAD BYTE AT (@TASK(0).PNTR);

ACTIVATE$TASK: PROCEDURE; /* ASSUMES NEW$TASK<>0 */
   DISABLE;
   IF (TASK(NEW$TASK).STATUS AND BUSY$BIT)<>0 THEN STATUS=0;
   ELSE /* SINCE TASK IS NOT BUSY */ DO;
      IF NEW$TASK < FIRST$LOW$PRIORITY$TASK THEN DO;
         IF HIGH$PRIORITY$TAIL<>0 THEN DO;
            TASK(HIGH$PRIORITY$TAIL).PNTR=NEW$TASK;
            END;
         ELSE /* SINCE HIGH$PRIORITY$TAIL=0 THEN */ DO;
            HIGH$PRIORITY$HEAD=NEW$TASK;
            END;
         HIGH$PRIORITY$TAIL=NEW$TASK;
         END;
      ELSE /* SINCE TASK IS LOW PRIORITY THEN */ DO;
         IF LOW$PRIORITY$TAIL<>0 THEN DO;
            TASK(LOW$PRIORITY$TAIL).PNTR=NEW$TASK;
            END;
         ELSE /* SINCE LOW$PRIORITY$TAIL=0 THEN */ DO;
            LOW$PRIORITY$HEAD=NEW$TASK;
            END;
         LOW$PRIORITY$TAIL=NEW$TASK;
         END;
      TASK(NEW$TASK).PNTR=0;
      TASK(NEW$TASK).STATUS=TASK(NEW$TASK).STATUS OR
            BUSY$BIT OR READY$BIT;
      STATUS=TASK(NEW$TASK).STATUS;
      END;
   NEW$TASK=0;
   RETURN;
   END ACTIVATE$TASK;
```

```
ACTIVATE$DELAY: PROCEDURE;/*ASSUMES NEW$TASK, NEW$DELAY<>0*/
   DECLARE POINTER$0 BYTE, POINTER$1 BYTE;
   DECLARE OLD$DIFFERENCE INTEGER, DIFFERENCE INTEGER;
   DISABLE;
   IF (TASK(NEW$TASK).STATUS AND BUSY$BIT)<>0 THEN STATUS=0;
   ELSE /* SINCE TASK IS NOT BUSY */ DO;
      DIFFERENCE=INT(NEW$DELAY);
      POINTER$0=DELAY$HEAD;
      POINTER$1=0;
      DO WHILE POINTER$0<>0 AND DIFFERENCE>0;
         OLD$DIFFERENCE=DIFFERENCE;
         DIFFERENCE=DIFFERENCE-INT(TASK(POINTER$0).DELAY);
         IF DIFFERENCE>0 THEN DO;
            POINTER$1=POINTER$0;
            POINTER$0=TASK(POINTER$1).PNTR;
            END;
         END;
      TASK(NEW$TASK).PNTR=POINTER$0;
      TASK(POINTER$1).PNTR=NEW$TASK;
      IF POINTER$0=0 THEN TASK(NEW$TASK).DELAY=LOW(UNSIGN(DIFFERENCE));
      ELSE /* SINCE DIFFERENCE<0 THEN */ DO;
         IF POINTER$0=POINTER$1 THEN TASK(POINTER$1).PNTR=0;
         TASK(NEW$TASK).DELAY=LOW(UNSIGN(OLD$DIFFERENCE));
         TASK(POINTER$0).DELAY=LOW(UNSIGN(-DIFFERENCE));
         END;
      TASK(NEW$TASK).STATUS=TASK(NEW$TASK).STATUS OR
            BUSY$BIT OR DELAY$BIT;
      STATUS=TASK(NEW$TASK).STATUS;
      END;
   NEW$TASK=0;
   NEW$DELAY=0;
   RETURN;
   END ACTIVATE$DELAY;

DECREMENT$DELAY: PROCEDURE;     /* ASSUMES INTERRUPTS DISABLED */
   DECLARE OFF$DELAY BYTE;
   IF DELAY$HEAD<>0 THEN DO;
      TASK(DELAY$HEAD).DELAY=TASK(DELAY$HEAD).DELAY-1;
      DO WHILE DELAY$HEAD<>0 AND TASK(DELAY$HEAD).DELAY=0;
         OFF$DELAY=DELAY$HEAD;
         DELAY$HEAD=TASK(DELAY$HEAD).PNTR;
         TASK(OFF$DELAY).STATUS=TASK(OFF$DELAY).STATUS
             AND NOT(BUSY$BIT OR DELAY$BIT);
         NEW$TASK=OFF$DELAY;
         CALL ACTIVATE$TASK;
         END;
      END;
   RETURN;
   END DECREMENT$DELAY;
```

```
CASE$TASK: PROCEDURE REENTRANT;
   DO CASE RUNNING$TASK;
      CALL TASK00;
      CALL TASK01;
      CALL TASK02;
      CALL TASK03;
      CALL TASK04;
      CALL TASK05;
      CALL TASK06;
      CALL TASK07;
      CALL TASK08;
      CALL TASK09;
      END;
   TASK(RUNNING$TASK).STATUS=TASK(RUNNING$TASK).STATUS AND
            NOT (BUSY$BIT OR READY$BIT);
   TASK(RUNNING$TASK).PNTR=0;
   IF RUNNING$TASK=NEW$TASK THEN DO;
      IF NEW$DELAY<>0 THEN DO;
         CALL ACTIVATE$DELAY;
         END;
      ELSE /* SINCE NEW$DELAY=0 */ DO;
         CALL ACTIVATE$TASK;
         END;
      END;
   RUNNING$TASK=0;
   RETURN;
   END CASE$TASK;


PREEMPT:PROCEDURE REENTRANT; /* ASSUMES INTERRUPTS DISABLED */
   IF PREEMPTED$TASK=0 THEN DO;
      IF (HIGH$PRIORITY$HEAD<>0) AND (RUNNING$TASK>=
            FIRST$LOW$PRIORITY$TASK) THEN DO;
         PREEMPTED$TASK=RUNNING$TASK;
         RUNNING$TASK=0;
         DO WHILE PREEMPTED$TASK<>0;
            CALL DISPATCH;
            END;
         END;
      END;
   RETURN;
   END PREEMPT;
```

```
DISPATCH:PROCEDURE REENTRANT; /* ASSUMES RUNNING$TASK=0 */
   DISABLE;
   IF HIGH$PRIORITY$HEAD<>0 THEN DO;
      RUNNING$TASK=HIGH$PRIORITY$HEAD;
      HIGH$PRIORITY$HEAD=TASK(RUNNING$TASK).PNTR;
      IF HIGH$PRIORITY$HEAD = 0 THEN HIGH$PRIORITY$TAIL = 0;
      CALL CASE$TASK;
      END;
   ELSE IF PREEMPTED$TASK<>0 THEN DO;
      RUNNING$TASK=PREEMPTED$TASK;
      PREEMPTED$TASK=0;
      END;
   ELSE IF LOW$PRIORITY$HEAD<>0 THEN DO;
      RUNNING$TASK=LOW$PRIORITY$HEAD;
      LOW$PRIORITY$HEAD=TASK(RUNNING$TASK).PNTR;
      IF LOW$PRIORITY$HEAD = 0 THEN LOW$PRIORITY$TAIL = 0;
      CALL CASE$TASK;
      END;
   ELSE RETURN;
   RETURN;
   END DISPATCH;
```

```
TASK00: PROCEDURE REENTRANT;/*ERROR CODE*/RETURN;END TASK00;

TASK01: PROCEDURE REENTRANT;
   ENABLE;
              /*TASK01 CODE*/
   DISABLE;
   RETURN;
   END TASK01;

TASK02: PROCEDURE REENTRANT;
   ENABLE;
              /*TASK02 CODE*/
   DISABLE;
   RETURN;
   END TASK02;

TASK03: PROCEDURE REENTRANT;
   ENABLE;
              /*TASK03 CODE*/
   DISABLE;
   RETURN;
   END TASK03;

TASK04: PROCEDURE REENTRANT;
   ENABLE;
              /*TASK04 CODE*/
   DISABLE;
   RETURN;
   END TASK04;

TASK05: PROCEDURE REENTRANT;
   ENABLE;
              /*TASK05 CODE*/
   DISABLE;
   RETURN;
   END TASK05;

TASK06: PROCEDURE REENTRANT;
   ENABLE;
              /*TASK06 CODE*/
   DISABLE;
   RETURN;
   END TASK06;

TASK07: PROCEDURE REENTRANT;
   ENABLE;
              /*TASK07 CODE*/
   DISABLE;
   RETURN;
   END TASK07;
```

```
TASKØ8: PROCEDURE REENTRANT;
   ENABLE;
               /*TASKØ8 CODE*/
   DISABLE;
   RETURN;
   END TASKØ8;

TASKØ9: PROCEDURE REENTRANT;
   ENABLE;
               /*TASKØ9 CODE*/
   DISABLE;
   RETURN;
   END TASKØ9;

               /*INITIALIZE*/

DISABLE;
DO STATUS=Ø TO 9;
   TASK(STATUS).PNTR=Ø;
   TASK(STATUS).STATUS=Ø;
   TASK(STATUS).DELAY=Ø;
   NEW$TASK,NEW$DELAY=Ø;
   HIGH$PRIORITY$HEAD,HIGH$PRIORITY$TAIL=Ø;
   LOW$PRIORITY$HEAD,LOW$PRIORITY$TAIL=Ø;
   RUNNING$TASK,PREEMPTED$TASK=Ø;
   END;

               /* MAIN LOOP */

DO WHILE TRUE<>FALSE;
   CALL DISPATCH;
   ENABLE;
   STATUS=STATUS;
   END;



END TM36;
```

REFERENCES

1. Hansen, Brinch, *Operating System Principles*, Prentice-Hall, Englewood, N.J., 1973.

2. Knuth, D. E., *The Art of Computer Programming*, Addison-Wesley, Reading, Mass., 1969.

3. Wirth, Nicklaus, *Algorithms + Data Structures = Programs*, Prentice-Hall, Englewood, N.J., 1976.

4. "PL/M-86 Programming Manual," Intel Corporation, 1978, manual order number 9800466A.

5. "RMX/80 User's Guide," Intel Corporation, 1977, manual order number 9800522B.

intel®

U.S. AND CANADIAN DISTRIBUTORS

3065 Bowers Avenue
Santa Clara, California 95051
Tel: (408) 987-8080
TWX: 910-338-0026
TELEX: 34-6372

June 1979

ALABAMA

†Hamilton/Avnet Electronics
4692 Commercial Drive N.W.
Huntsville 35805
Tel: (205) 837-7210

Pioneer
1207 Putman Drive NW
Huntsville 35805
Tel: (205) 837-9033
TWX: 810-726-2197

ARIZONA

†Hamilton/Avnet Electronics
2615 South 21st Street
Phoenix 85034
Tel: (602) 275-7851

†Liberty/Arizona
8155 N. 24th Avenue
Phoenix 85021
Tel: (602) 249-2232
TWX: 910-951-4282

CALIFORNIA

†Avnet Electronics
350 McCormick Avenue
Costa Mesa 92626
Tel: (714) 754-6111

Hamilton/Avnet
1175 Bordeaux Dr.
Sunnyvale 94086
Tel: (408) 743-3300
TWX: 910-339-9332

†Hamilton/Avnet Electronics
8917 Complex Drive
San Diego 92123
Tel: (714) 279-2421
TWX: 910-335-1216

Hamilton/Avnet
10912 W. Washington Blvd.
Culver City 90230
Tel: (213) 558-2809 (2665)
TWX: 910-340-6364 or 7073

†Hamilton Electro Sales
10912 W. Washington Boulevard
Culver City 90230
Tel: (213) 558-2121

†Liberty Electronics
124 Maryland Street
El Segundo 90245
Tel. (213) 322-3826
TWX: 910-348-7140 or 7111

†Liberty/San Diego
9525 Chesapeake Dr.
San Diego 92123
Tel: (714) 565-9171
TWX: 910-335-1590

†Elmar Electronics
3000 Bowers Avenue
Santa Clara 95052
Tel: (408) 727-2500
TWX: 910-338-0451 or 0296

Hamilton/Avnet Electronics
17312 Eastman Street
Irvine 92714
Tel: (714) 979-6864

COLORADO

†Elmar/Denver
6777 E. 50th Avenue
Commerce City 80022
Tel: (303) 287-9611
TWX: 910-931-0510

†Hamilton/Avnet Electronics
5921 No. Broadway
Denver 80216
Tel: (303) 534-1212
TWX: 910-931-0510

CONNECTICUT

†Cramer/Connecticut
P.O. Box 5003
12 Beaumont Road
Wallingford 06492
Tel: (203) 265-7741
TWX: 710-476-0162

†Hamilton/Avnet Electronics
643 Danbury Road
Georgetown 06829
Tel: (203 762-0361

†Harvey Electronics
112 Main Street
Norwalk 06851
Tel: (203) 853-1515
TWX: 710-468-3373

FLORIDA

Arrow Electronics
1001 N.W. 62nd Street
Suite 108
Ft. Lauderdale 33309
Tel: (305) 776-7790

Arrow Electronics
115 Palm Bay Road, NW
Suite 10, Bldg. 200
Palm Bay 32905
Tel: (305) 725-1480
TWX: 510-959-6337

†Hamilton/Avnet Electronics
6800 Northwest 20th Ave.
Ft. Lauderdale 33309
Tel: (305) 971-2900
TWX: 510-955-3097

†Pioneer
6220 S. Orange Blossom Trail
Suite 412
Orlando 32809
Tel: (305) 859-3600
TWX: 810-850-0177

Hamilton/Avnet
3197 Tech. Drive N.
St. Petersburg 33702
Tel: (813) 576-3930
TWX: 810-863-0374

GEORGIA

Arrow Electronics
3406 Oak Cliff Road
Doraville 30340
Tel: (404) 455-4054
TWX: 810-757-4213

†Hamilton/Avnet Electronics
6700 i-85 Access Road, #11
Norcross 30071
Tel: (404) 448-0800

ILLINOIS

Arrow Electronics
492 Lunt Avenue
P.O. Box 94248
Schaumburg 60193
Tel: (312) 593-8230
TWX: 910-222-1807

†Hamilton/Avnet Electronics
3901 No. 25th Ave.
Schiller Park 60176
Tel: (317) 678-6310
TWX: 910-227-0060

Pioneer/Chicago
1551 Carmen Drive
Elk Grove Village 60006
Tel: (312) 437-9680
TWX: 910-222-1834

INDIANA

†Pioneer/Indiana
6408 Castleplace Drive
Indianapolis 46250
Tel: (317) 849-7300
TWX: 810-260-1794

KANSAS

†Hamilton/Avnet Electronics
9219 Quivira Road
Overland Park 66215
Tel: (913) 888-8900

MARYLAND

†Hamilton Avnet
P.O. Box 647,
BWI Airport
7235 Standard Drive
Hanover 21076
Tel: (301) 796-5684
TWX: 710-862-1861

†Pioneer/Washington
9100 Gaither Road
Gaithersburg 20760
Tel: (301) 948-0710
TWX: 710-828-0545

MASSACHUSETTS

†Cramer Electronics Inc.
85 Wells Avenue
Newton 02159
Tel: (617) 969-7700

MASSACHUSETTS (continued)

†Hamilton/Avnet Electronics
50 Tower Office Park
Woburn 01801
Tel: (617) 273-7500

MICHIGAN

†Arrow Electronics
3921 Varsity Road
Ann Arbor 48140
Tel: (313) 971-8220
TWX: 810-223-6020

†Pioneer/Michigan
13485 Stamford
Livonia 48150
Tel: (313) 525-1800
TWX: 810-242-3271

†Hamilton/Avnet Electronics
32487 Schoolcraft Road
Livonia 48150
Tel: (313) 522-4700
TWX: 810-242-8775

MINNESOTA

†Industrial Components
5280 West 74th Street
Minneapolis 55435
Tel: (612) 831-2666
TWX: 910-576-3153

Arrow Electronics
5251 73rd Street
Edina 55435
Tel: (612) 835-7811
TWX: 910-576-2726

†Hamilton/Avnet Electronics
7449 Cahill Road
Edina 55435
Tel. (612) 941-3801
TWX: 910-576-2720

MISSOURI

†Hamilton/Avnet Electronics
396 Brookes Drive
Hazelwood 63042
Tel: (314) 731-1144
TWX: 910-762-0606

NEW JERSEY

Arrow/Philadelphia
Pleasant Valley
Moorestown 08057
Tel: (201) 239-0800
TWX: 710-897-0829

Arrow Electronics
285 Midland Avenue
Saddlebrook 07662
Tel. (201) 797-5800
TWX. 710-988-2206

Hamilton/Avnet
10 Industrial
Fairfield 07006
Tel: (201) 575-3390
TWX: 710-734-4338

†Harvey Electronics
45 Route 46
Pinebrook 07058
Tel: (201) 227-1262
TWX: 710-734-4382

†Hamilton/Avnet Electronics
113 Gaither Drive
East Gate Industrial Park
Mt. Laurel 08057
Tel: (609) 424-0100
TWX: 710-897-1405

NEW MEXICO

Alliance Electronics Inc.
11721 Central Ave.
Albuquerque 87123
Tel: (505) 292-3360
TWX: 910-989-1151

†Hamilton/Avnet Electronics
2524 Baylor Drive, S.E.
Albuquerque 87119
Tel. (505) 765-1500

†Microcomputer System Technical Demonstrator Centers

intel®

U.S. AND CANADIAN DISTRIBUTORS

3065 Bowers Avenue
Santa Clara, California 95051
Tel: (408) 987-8080
TWX: 910-338-0026
TELEX: 34-6372

June 1979

NEW YORK

Harvey Electronics
P.O. Box 1208
Binghampton 13902
Tel: (607) 748-8211
TWX: 510-252-0893

Arrow Electronics
900 Broad Hollow Road
Farmingdale 11735
Tel: (516) 694-6800
TWX: 510-224-6494

·Cramer/Rochester
3000 South Winton Road
Rochester 14623
Tel: (716) 275-0300
TWX: 910-338-0026

‡Hamilton/Avnet Electronics
167 Clay Road
Rochester 14623
Tel: (716) 442-7820
TWX: 910-340-6364

†Cramer/Syracuse
7705 Maltlage Drive
Liverpool 13088
Tel: (315) 652-1000
TWX: 710-545-0230

Arrow Electronics
399 Conklin Street
Farmingdale 11735
Tel: (516) 694-6800
TWX: 510-224-6494

‡Hamilton/Avnet Electronics
16 Corporate Circle
E. Syracuse 13057
Tel: (315) 437-2641

‡Hamilton/Avnet Electronics
70 State Street
Westbury, L.I. 11590
Tel: (516) 333-5413
TWX: 510-252-0893

‡Harvey Electronics
60 Crossways Park West
Woodbury 11797
Tel: (516) 921-8700
TWX: 510-221-2184

NORTH CAROLINA

Pioneer/Carolina
106 Industrial Ave.
Greensboro 27406
Tel: (919) 273-4441
TWX: 510-925-1114

‡Hamilton/Avnet Electronics
2803 Industrial Drive
Raleigh 27609
Tel: (919) 829-8030

Arrow Electronics
P.O. Box 989
Kernersville 27284
Tel: (919) 996-2039
TWX: 510-922-4765

OHIO

Arrow Electronics
3100 Plainfield Road
Kettering 45432
Tel: (513) 253-9176
TWX: 810-459-1611

Arrow Electronics
6238 Cochran Rd.
Solon 44139
Tel: (216) 248-3990

‡Hamilton/Avnet Electronics
954 Senate Drive
Dayton 45459
Tel: (513) 433-0610
TWX: 910-340-2531

‡Pioneer/Dayton
1900 Troy Street
Dayton 45404
Tel: (513) 236-9900
TWX: 810-459-1622

Arrow Electronics
P.O. Box 37856
Cincinnati 45222
Tel: (513) 761-5432
TWX: 810-422-2210

‡Pioneer/Cleveland
4800 E. 131st Street
Cleveland 44105
Tel: (216) 587-3600
TWX: 810-422-2210

OHIO (continued)

†Hamilton/Avnet Electronics
4588 Emory Industrial Parkway
Warrensville Heights 44128
Tel: (216) 831-3500

OKLAHOMA

·Components Specialties, Inc.
7920 E. 40th Street
Tulsa 74145
Tel: (918) 664-2820
TWX: 910-845-2215

OREGON

‡Almac/Stroum Electronics
8022 S.W. Nimbos, Bldg. 7
Beaverton 97005
Tel: (503) 641-9070

PENNSYLVANIA

Arrow Electronics
4297 Greensberg Pike
Suite 3114
Pittsburgh 15221
Tel: (412) 351-4000

Pioneer/Pittsburgh
560 Alpha Drive
Pittsburgh 15238
Tel: (412) 782-2300
TWX: 710-795-3122

Pioneer/Delaware Valley
141 Gibraltar Road
Horsham 19044
Tel: (215) 674-4000
TWX: 510-665-6778

TENNESSEE

Arrow Electronics
6900 Office Park Circle
Knoxville 37919
Tel: (615) 588-5836

TEXAS

Component Specialties Inc.
8222 Jamestown Drive
Suite 115
Austin 78758
Tel: (512) 837-8922
TWX: 910-874-1320

†Hamilton/Avnet Electronics
4445 Sigma Road
Dallas 75240
Tel: (214) 661-8661
TWX: 910-860-5371

†Hamilton/Avnet Electronics
3939 Ann Arbor
Houston 77063
Tel: (713) 780-1771

‡Component Specialties, Inc.
10807 Shady Trail, Suite 101
Dallas 75220
Tel: (214) 357-6511
TWX: 910-861-4999

†Component Specialties, Inc.
8585 Commerce Park Drive, Suite 590
Houston 77036
Tel: (713) 771-7237
TWX: 910-881-2422

Arrow Electronics
13715 Gamma Road
Dallas 75234
Tel: (214) 661-9300
TWX: 910-861-5495

UTAH

†Hamilton/Avnet Electronics
1585 West 2100 South
Salt Lake City, 84119
Tel: (801) 972-2800

WASHINGTON

‡Hamilton/Avnet Electronics
14212 N.E. 21st
Bellevue 98005
Tel: (206) 746-8750

‡Almac/Stroum Electronics
5811 Sixth Ave. South
Seattle 98108
Tel: (206) 763-2300
TWX: 910-444-2067

WASHINGTON (continued)

‡Liberty Electronics
1750 132nd Avenue NE
Bellevue 98005
Tel: (206) 453-8300
TWX: 910-443-2526

WISCONSIN

Arrow Electronics
434 W. Rausson Avenue
Oak Creek 53154
Tel: (414) 764-6600
TWX: 910-338-0026

†Hamilton/Avnet
2975 Moorland Road
New Berlin 53151
Tel: (414) 784-4510
TWX: 910-262-1182

CANADA

ALBERTA

‡L. A. Varah Ltd.
4742 14th Street N.E.
Calgary T2E 6L7
Tel: (403) 230-1235
TWX: 018-258-97

BRITISH COLUMBIA

‡L.A. Varah Ltd.
2077 Alberta Street
Vancouver V5Y 1C4
Tel: (604) 873-3211
TWX: 610-929-1068

Zentronics
8325 Fraser Street
Vancouver V5X 3X8
Tel: (604) 325-3292
TWX: 04-5077-89

MANITOBA

L. A. Varah
1-1832 King Edward Street
Winnipeg R2R 0N1
Tel: (204) 633-6190
TWX: 07-55-365

ONTARIO

‡L.A. Varah, Ltd.
505 Kenora Avenue
Hamilton L8E-3P2
Tel: (416) 561-9311
TWX: 061-8349

†Hamilton/Avnet Electronics
3688 Nashua Drive, Units G & H
Mississauga L4V 1M5
Tel: (416) 677-7432
TWX: 610-492-8860

‡Hamilton/Avnet Electronics
1735 Courtwood Cresc.
Ottawa K2C 3J2
Tel: (613) 226-1700

‡Zentronics
141 Catherine Street
Ottawa, Ontario K2P 1C3
Tel: (613) 238-6411
TWX: 053-3636

‡Zentronics
1355 Meyerside Drive
Mississauga, Ontario L5T 1C9
Tel: (416) 676-9000
Telex: 06-983-657

QUEBEC

‡Hamilton/Avnet Electronics
2670 Paulus Street
St. Laurent H4S 1G2
Tel: (514) 331-3731
TWX: 610-421-3731

Zentronics
5010 Pare Street
Montreal H4P 1P3
Tel: (514) 735-5361
TWX: 05-827-535

‡Microcomputer System Technical Demonstrator Centers

U.S. AND CANADIAN SALES OFFICES

3065 Bowers Avenue
Santa Clara, California 95051
Tel: (408) 987-8080
TWX: 910-338-0026
TELEX: 34-6372

June 1979

ALABAMA
Intel Corp.
3322 S. Parkway, Ste. 71
Holiday Office Center
Huntsville 35802
Tel: (205) 883-2430

Glen White Associates
3502 9th Avenue
Huntsville 35805
Tel: (205) 533-5272

†Pen-Tech Associates, Inc.
Holiday Office Center
3322 S. Memorial Pkwy.
Huntsville 35801
Tel: (205) 533-0090

ARIZONA
Intel Corp.
8650 N. 35th Avenue, Suite 101
Phoenix 85021
Tel. (602) 242-7205

†BFA
4426 North Saddle Bag Trail
Scottsdale 85251
Tel: (602) 994-5400

CALIFORNIA
Intel Corp.
7670 Opportunity Rd.
Suite 135
San Diego 92111
Tel: (714) 268-3563

Intel Corp.*
1651 East 4th Street
Suite 105
Santa Ana 92701
Tel: (714) 835-9642
TWX: 910-595-1114

Intel Corp.*
15335 Morrison
Suite 345
Sherman Oaks 91403
(213) 986-9510
TWX: 910-495-2045

Intel Corp.*
3375 Scott Blvd.
Santa Clara 95051
Tel. (408) 987-8086
TWX: 910-339-9279
TWX: 910-338-0255

Earle Associates, Inc.
4617 Ruffner Street
Suite 202
San Diego 92111
Tel: (714) 278-5441

Mac-I
2576 Shattuck Ave.
Suite 4B
Berkeley 94704
Tel: (415) 843-7625

Mac-I
P.O. Box 1420
Cupertino 95014
Tel: (408) 257-9880

Mac-I
P.O. Box 8763
Fountain Valley 92708
Tel: (714) 839-3341

Mac-I
20121 Ventura Blvd., Suite 240E
Woodland Hills 91364
Tel: (213) 347-5900

COLORADO
Intel Corp.*
6000 East Evans Ave
Bldg. 1, Suite 260
Denver 80222
Tel: (303) 758-8086
TWX: 910-931-2289

‡Westek Data Products, Inc
27972 Meadow Drive
P.O. Box 1355
Evergreen 80439
Tel. (303) 674-5255

Westek Data Products, Inc.
1322 Arapahoe
Boulder 80302
Tel. (303) 449-2620

CONNECTICUT
Intel Corp.
Peacock Alley
1 Padanaram Road, Suite 146
Danbury 06810
Tel. (203) 792-8366
TWX: 710-456-1199

FLORIDA
Intel Corp.
1001 N.W. 62nd Street, Suite 406
Ft. Lauderdale 33309
Tel. (305) 771-0600
TWX: 510-956-9407

Intel Corp.
5151 Adanson Street, Suite 203
Orlando 32804
Tel. (305) 628-2393
TWX: 810-853-9219

FLORIDA (cont.)
‡Pen-Tech Associates, Inc.
201 S.E. 15th Terrace, Suite F
Deerfield Beach 33441
Tel: (305) 421-4989

‡Pen-Tech Associates, Inc.
111 So. Maitland Ave., Suite 202
Maitland 32751
Tel: (305) 645-3444

GEORGIA
‡Pen-Tech Associates, Inc.
Suite 305 C
2101 Powers Ferry Road
Atlanta 30339
Tel: (404) 955-0293

ILLINOIS
Intel Corp.*
900 Jorie Boulevard
Suite 220
Oakbrook 60521
Tel. (312) 325-9510
TWX: 910-651-5881

First Rep Company
9400-9420 W. Foster Avenue
Chicago 60656
Tel. (312) 992-0830

INDIANA
Electro Reps Inc.
941 E. 86th Street, Suite 101
Indianapolis 46240
Tel: (317) 255-4147
TWX: 810-341-3217

Electro Reps Inc.
3601 Hobson Rd
Suite 106
Ft. Wayne 46815
Tel. (219) 483-0518

IOWA
Technical Representatives, Inc
St. Andrews Building
1930 St. Andrews Drive N.E
Cedar Rapids 52405
Tel. (319) 393-5510

KANSAS
Intel Corp.
9393 W. 110th St., Ste. 265
Overland Park 66210
Tel. (913) 642-8080

Technical Representatives, Inc
8245 Nieman Road, Suite 114
Lenexa 66214
Tel. (913) 888-0212, 3, & 4
TWX: 910-749-6412

KENTUCKY
‡Lowry & Associates, Inc.
3351 Commodore
Lexington 40502
Tel: (606) 269-6329

MARYLAND
Intel Corp.*
7257 Parkway Drive
Hanover 21076
Tel: (301) 796-7500
TWX: 710-862-1944

Glen White Associates
57 W. Timonium Road, Suite 307
Timonium 21093
Tel: (301) 252-6360

†Mesa Inc.
11900 Parklawn Drive
Rockville 20852
Tel. Wash. (301) 881-8430
Balto. (301) 792-0021

MASSACHUSETTS
Intel Corp.*
27 Industrial Ave.
Chelmsford 01824
Tel: (617) 667-8126
TWX: 710-343-6333

†Computer Marketing, Inc.
257 Crescent Street
Waltham 02154
Tel. (617) 894-7000

MICHIGAN
Intel Corp.*
26500 Northwestern Hwy
Suite 401
Southfield 48075
Tel. (313) 353-0920
TWX: 910-420-1212
TELEX: 2 31143

†Lowry & Associates, Inc.
135 W. North Street
Suite 4
Brighton 48116
Tel. (313) 227-7067

MINNESOTA
Intel Corp
8200 Normandale Avenue
Suite 422
Bloomington 55437
Tel. (612) 835-6722
TWX: 910-576-2867

†Dytek North
1821 University Ave.
Room 163N
St. Paul 55104
Tel: (612) 645-5816

MISSOURI
Technical Representatives Inc.
320 Brookes Drive, Suite 104
Hazelwood 63042
Tel. (314) 731-5200
TWX: 910-762-0618

NEW JERSEY
Intel Corp.*
1 Metroplaza Office Bldg.
505 Thornall St.
Edison 08817
Tel: (201) 494-5040
TWX: 710-480 6238

NEW MEXICO
BFA Corporation
P.O. Box 1237
Las Cruces 88001
Tel: (505) 523-0601
TWX: 910-983-0543

BFA Corporation
3705 Westerfield, N.E.
Albuquerque 87111
Tel. (505) 292-1212
TWX: 910-989-1157

NEW YORK
Intel Corp.*
350 Vanderbilt Motor Pkwy
Suite 402
Hauppauge 11787
Tel. (516) 231-3300
TWX: 510-227-6236

Intel Corp.
80 Washington St.
Poughkeepsie 12601
Tel. (914) 473-2303
TWX: 510 248 0060

Intel Corp.
2255 Lyell Avenue
Lower Floor East Suite
Rochester 14606
Tel. (716) 328-7340
TWX: 510-253-3841

‡Measurement Technology, Inc.
159 Northern Boulevard
Great Neck 11021
Tel: (516) 482-3500

T-Squared
4054 Newcourt Avenue
Syracuse 13206
Tel. (315) 463-8592
TWX: 710-541-0554

T-Squared
2 E. Main
Victor 14564
Tel. (716) 924-9101
TWX: 510-254-8542

NORTH CAROLINA
‡Pen-Tech Associates, Inc.
1202 Eastchester Dr.
Highpoint 27260
Tel: (919) 883-9125

Glen White Associates
4021 Barrett Dr.
Suite 12
Raleigh 27609
Tel: (919) 787-7016

OHIO
Intel Corp.*
8312 North Main Street
Dayton 45415
Tel. (513) 890-5350
TWX: 810-450-2528

Intel Corp.*
Chagrin-Brainard Bldg. #201
28001 Chagrin Blvd.
Cleveland 44122
Tel: (216) 464-2736

Lowry & Associates, Inc.
24200 Chagrin Blvd.
Suite 320
Cleveland 44122
Tel: (216) 464-8113

‡Lowry & Associates, Inc.
1524 Marsetta Drive
Dayton 45432
Tel. (513) 429-9040

‡Lowry & Associates, Inc.
1050 Freeway Dr., N.
Suite 209
Columbus 43229
Tel. (614) 436-2051

OREGON
Intel Corp.
10700 S.W. Beaverton
Hillsdale Highway
Suite 324
Beaverton 97005
Tel: (503) 641-8086

E.S. Chase Company
4095 SW 144th St.
Beaverton 97005
Tel. (503) 641-4111

PENNSYLVANIA
Intel Corp.*
275 Commerce Dr.
200 Office Center
Suite 300
Fort Washington 19034
Tel. (215) 542-9444
TWX: 510-661-2077

‡Lowry & Associates, Inc.
Seven Parkway Center
Suite 455
Pittsburgh 15220
Tel. (412) 922-5110

‡Q.E.D. Electronics
300 N. York Road
Hatboro 19040
Tel. (215) 674-9600

TENNESSEE
Glen White Associates
Rt. #12, Norwood S.D.
Jonesboro 37659
Tel. (615) 477-8850

Glen White Associates
2523 Howard Road
Germantown 38138
Tel: (901) 754-0483

Glen White Associates
6446 Ridge Lake Road
Hixon 37343
Tel. (615) 842-7799

TEXAS
Intel Corp.*
2925 L.B.J. Freeway
Suite 175
Dallas 75234
Tel. (214) 241-9521
TWX: 910 860-5487

Intel Corp.*
6776 S.W. Freeway
Suite 550
Houston 77074
Tel: (713) 784-3400

Mycrosystems Marketing Inc
13777 N. Central Expressway
Suite 405
Dallas 75243
Tel. (214) 238 7157
TWX: 910-867-4763

Mycrosystems Marketing Inc
6610 Harwin Avenue, Suite 125
Houston 77036
Tel. (713) 783-2900

Mycrosystems Marketing Inc
Koger Executive Center
Suite 207
San Antonio 78228
Tel. (512) 735 5073

VIRGINIA
Glen White Associates
P.O. Box 1104
Lynchburg 24505
Tel. (804) 384-6920

Glen White Associates
Rt. #1 Box 322
Colonial Beach 22443
Tel. (804) 224-4871

WASHINGTON
Intel Corp.
Campus Office Park Bldg. 3
1603 116th Ave. N.E.
Bellevue 98005
Tel. (206) 453-8086

E.S. Chase Co.
P.O. Box 80903
Seattle 98108
Tel. (206) 762 4824
TWX: 910-444 2298

WISCONSIN
Intel Corp.
4369 S. Howell Ave.
Milwaukee 53207
Tel. (414) 747-0789

CANADA
Intel Semiconductor Corp.*
Suite 233 Bell Mews
39 Highway 7, Bells Corners
Ottawa, Ontario K2H 8R2
Tel. (613) 829-9714
TELEX: 053-4115

Intel Semiconductor Corp.
6205 Airport Rd.
Bldg. B Suite 205
Mississauga Ontario
L4V 1E3
Tel. (416) 671-0611
TELEX: 06983574

Multilek Inc.*
15 Grenfell Crescent
Ottawa Ontario K2G 0G3
Tel. (613) 226-2365
TELEX: 053-4585

*Field application location
‡These representatives do not offer Intel Components, only boards and systems

INTERNATIONAL SALES AND MARKETING OFFICES

3065 Bowers Avenue
Santa Clara, California 95051
Tel. (408) 987 8080
TWX: 910 338-0026
TELEX: 34-6372

June 1979

EUROPEAN MARKETING OFFICES

BELGIUM
Intel Corporation, S.A.*
Rue du Moulin a Papier 51
Boite 1
B-1160 Brussels
Tel. (02) 660 30 10
TELEX. 24814

DENMARK
Intel Denmark A/S*
Lyngbyvej 32 2nd Floor
DK 2100 Copenhagen East
Tel. (01) 18 20 00
TELEX 19567

ENGLAND
Intel Corporation (U.K.) Ltd.*
Broadfield House
4 Between Towns Road
Cowley, Oxford OX4 3NB
Tel. (0865) 77 14 31
TELEX 837203

Intel Corporation (U.K.) Ltd.
5 Hospital Street
Nantwich, Cheshire CW5 5RE
Tel. (0270) 62 65 60
TELEX 36620

FINLAND
Intel Sweden AB
P.O. Box 17
Sentnerikuja, 3
SF-00400
Helsinki, Finland
Tel. 358 0/55 85 31
TELEX 123332

FRANCE
Intel Corporation, S.A.R.L.*
5 Place de la Balance
Silic 223
94528 Rungis Cedex
Tel. (01) 687 22 21
TELEX 270475

GERMANY
Intel Semiconductor GmbH*
Seidlstrasse 27
8000 Muenchen 2
Tel. (089) 55 87 41
TELEX. 523 177

Intel Semiconductor GmbH
Abraham Lincoln Strasse 30
6200 Wiesbaden 1
Tel. (06121) 74855
TELEX. 04186183

Intel Semiconductor GmbH
Werner Strasse 67
P.O. Box 1460
7012 Fellbach
Tel: (0711) 580082
TELEX 7254826

Intel Semiconductor GmbH
Hindenburger Strasse 28/29
3000 Hannover
Tel. (0511) 652051
TELEX. 923625

ISRAEL
Intel Semiconductor Ltd.*
P.O. Box 1659
Haifa
Tel: 972/452 4261
TELEX 92246511

ITALY
Intel Corporation Italia, s.p.a.
Corso Sempione 39
I-20145 Milano
Tel. 39/2/34.93 18
TELEX. 311271

NETHERLANDS
Intel Semiconductor Nederland B.V.
Cometongebouw
Westblaak 106
3012 Km Rotterdam
Tel: (10) 149122
TELEX: 22283

NORWAY
Intel Norway A/S
P.O. Box 158
N-2040
Kloita, Norway
Tel. 47 2/981068
TELEX: 18018

SWEDEN
Intel Sweden AB*
Box 20092
Enighetsvagen, 5
S-16120 Bromma
Sweden
Tel: (08) 98 53 90
TELEX: 12261

SWITZERLAND
Intel Semiconductor AG
Forchastrasse 95
CH 8035 Zurich
Tel. 1-55 45 02

ORIENT MARKETING OFFICE

JAPAN
Intel Japan K.K.*
Flower Hill-Shinmachi East Bldg.
1-23-9, Shinmachi, Setagaya-ku
Tokyo 154
Tel. (03) 426-9261
TELEX. 781-28426

HONG KONG
Intel Trading Corporation
99-105 Des Voeux Rd., Central
18F Unit B
Hong Kong

INTERNATIONAL DISTRIBUTORS/REPRESENTATIVES

AUSTRALIA
A.J.F. Systems & Components
PTY. LTD.
44 Prospect Rd
Prospect
South Australia 5082
Tel. 269-1244
TELEX 82635

A.J.F. Systems & Components
PTY. LTD.
29 Devlin St
Ryde N.S.W.
Tel. 807-6878
TELEG. 24906

A.J.F. Systems & Components
PTY. LTD.
310 Queen St. Melbourne,
Victoria 3000
Tel. 678-702
TELEX. 30270

Warburton Franki (Sydney) Pty. Ltd.
199 Parramatta Road
Auburn, N.S.W. 2144
Tel. 648-1711, 648-1381
TELEX. WARFRAN AA 22265

Warburton Franki Industries
(Melbourne) Pty. Ltd.
220 Park Street
South Melbourne, Victoria 3205
Tel. 699-4999
TELEX. WARFRAN AA 31370

Warburton Franki, Pty. Ltd.
322 Grange Road, Kidman Park
South Australia 5025
Tel. 356-7333
TELEX. WARFRAN AA 92908

Warburton Franki (Perth) Pty. Ltd.
98-102 Belgravia St., Belmont
Western Australia 6104
Tel. 356-7000
TELEX. WARFRAN AA 92908

Warburton Franki (Brisbane) Pty. Ltd.
13 Chester St., Fortitude Valley
Queensland 4006
TELEX. WARFRAN AA 41052

Warburton O'Donnell Limited
Corporate Headquarters
372 Eastern Valley Way
Chatswood, New South Wales 2067
Tel. 407-3261
TELEX. AA 21299

AUSTRIA
Bacher Elektronische Geraete GmbH
Rotenmulgasse 26
A 1120 Vienna
Tel. (0222) 83 63 96
TELEX (011) 1532

Rekirsch Elektronik Geraete GmbH
Lichtensteinstrasse 97
A1090 Vienna
Tel. (222) 347646
TELEX 74759

BELGIUM
Inelco Belgium S.A.
Avenue Val Duchesse, 3
B-1160 Brussels
Tel. (02) 660 00 12
TELEX 25441

BRAZIL
Icotron S.A.
05110-Av. Mutinga 3650
6 Andar
Pirituba-Sao Paulo
Tel. 261-0211
TELEX (011) 222 ICO BR

COLOMBIA
International Computer Machines
Diagonal 34, No. 5-62
Apartado Aereo 27599
Bogota
Tel. 232-6635
TELEX. 43439

DENMARK
Lyngso Komponent A.S.
Ostmarken 4
DK-2860 Soborg
Tel. (01) 67 00 77
TELEX 22990

Scandinavian Semiconductor
Supply A.S.
Nannasgade 18
DK-2200 Copenhagen N
Tel: (01) 83 50 90
TELEX 19037

FINLAND
Oy Fintronic AB
Loennrotinkatu 35D
SF 00180
Helsinki 18
Tel. (80) 601155
TELEX. 123107

Oy Softplan AB
Erottajankatu 9A
SF-00100 Helsinki, 10
Tel. 80-64306
TELEX 12579

FRANCE
Celdis
53, Rue Charles Frerot
94250 Gentilly
Tel: 581 00 20/581 04 69
TELEX. 200 485 F

Metrologie
La Tour d'Asnieres
4, Avenue Laurent Cely
92606-Asnieres
Tel: 791 44 44
TELEX. 611 448 F

Tekelec Airtronic*
Cite des Bruyeres
Rue Carle Vernet
92310 Sevres
Tel. (1) 027 75 35
TELEX: 204552

GERMANY
Alfred Neye Enatechnik GmbH
Schillerstrasse 14
D-2085 Quickborn-Hamburg
Tel. (04106) 6121
TELEX 02 13590

Electronic 2000 Vertriebs GmbH
Neumarkter Strasse 75
D-8000 Muenchen 80
Tel. (089) 434061
TELEX. 522561

Jermyn GmbH
Postfach 1180
D-6277 Kamberg
Tel: (06434) 6005
TELEX: 484426

Mania
Hauptstrasse 86
D-6384 Schmitten 2
Tel. (6082) 2543
TELEX 415325

HONG KONG
Schmidt & Co.
28/F Wing on Center
Connaught Road
Hong Kong
Tel. 5-455-644
TELEX. 74766 Schmc Hx

INDIA
Micro Electronics International
10-2-289/114A
Shantinager
Hyderabad 500028
CABLE: MELECTRO-HYDERBAD

ISRAEL
Eastronics Ltd.*
11 Rozanis Street
P.O. Box 39300
Tel-Aviv 61390
Tel. 475151
TELEX: 33638

ITALY
Eledra 3S S.P.A.*
Viale Elvezia 18
20154 Milan,
Tel. (02) 3493041
TELEX: 39332

Eledra 3S S.P.A.*
Via Paolo Galdano, 141 D
10137 Torino
TEL. (011) 30 97 097 - 30 97 114
TELEX. 210632

Eledra 3S S.P.A.*
Via Giuseppe Valmarana, 63
00139 Rome, Italy
Tel. (06) 81 27 290 - 81 27 324
TELEX: 612051

JAPAN
Tokyo Electron Labs. Inc
No. 1 Higashikata-Machi
Midori-Ku, Yokohama 226
Tel. (045) 471-8811
TELEX. 781-4773

Ryoyo Electric Corp
Konwa Bldg.
1-12-22, Tsukiji, 1-Chome
Chuo-Ku, Tokyo 104
Tel: (03) 543-7711

Nippon Micro Computer Co. Ltd.
Mutsumi Bldg. 4-5-21 Kojimachi
Chiyoda-ku Tokyo 102
Tel. (03) 230-0041

Asahi Electronics Co. Ltd
KMM Bldg. Room 407
2-14-1 Asano, Kokura
Kita-Ku, Kitokyushu City 802
Tel: (093) 511-6471
TELEX. AECKY 7126-16

KOREA
Koram Digital
Room 411 Ahil Bldg
49-4 2-GA Hoehyun-Dong
Chung-Ku Seoul
Tel: 23-8123
TELEX: HANSINT K23542

Leewood International, Inc.
C.P.O. Box 4046
112-25, Sokong-Dong
Chung-Ku, Seoul 100
Tel: 28-5927
CABLE: "LEEWOOD" Seoul

NETHERLANDS
C.N. Rood BV
Cort Vender
Lindenstraat, 13
Postbus 42
Rijswijk 2280 AA
Tel: 070-996360
TELEX: 31238

Inelco Nederland
AFD Elektronic
Joan Muyskenweg 22
NL-1096 Amsterdam
Tel. (020) 934824
TELEX. 14622

NEW ZEALAND
W. K. McLean Ltd
103-5 Felton Mathew Avenue
Glenn Innes, Auckland, 6
Tel 587-037
TELEX NZ2763 KOSFY

NORWAY
Nordisk Elektronik (Norge) A.S.
Mustads Vei 1
N-Oslo 2
Tel: (02) 55 24 85
TELEX 16963

PORTUGAL
Ditram
Componentes E Electronica LDA
Av. Miguel Bombarda, 133
Lisboa 1
Tel. 119 45 313

SINGAPORE
General Engineers Assocrates
Blk 3, 1003-1008, 10th Floor
P.S.A. Multi-Storey Complex
Telok Blangah/Pasir Panjang
Singapore 5
Tel. 271-3163
TELEX RS23987 GENERCO

SOUTH AFRICA
Electronic Building Elements
Pine Square
18th Street
Hazelwood, Pretoria
Tel: 78 92 21
TELEX: 30181SA

SPAIN
Interface*
Ronda San Pedro 22
Barcelona 10
Tel. 301 78 51
TELEX: 51508 IFCE E

ITT SESA
Miguel Angel 16
Madrid 10
Tel: (1) 410 2354
TELEX: 27707/27461

SWEDEN
Nordisk Electronik Ab
Sandhamnsgatan 71
S-102 54 Stockholm
Tel. (08) 635040
TELEX 10547

SWITZERLAND
Industrade AG
Gemsenstrasse 2
Postcheck 80 - 21190
CH-8021 Zurich
Tel. (01) 60 22 30
TELEX: 56788

TAIWAN
Taiwan Automation Co.*
2nd Floor
224 Nanking East Road
Section 3
Taipei
Tel. 771-0949
TELEX: 11942 TAIAUTO

UNITED KINGDOM
G.E.C. Semiconductors Ltd.
East Lane
North Wembley
Middlesex HA9 7PP
Tel. (01) 904-9303/908-4111
TELEX: 923492

Jermyn Industries
Vestry Estate
Sevenoaks, Kent
Tel: (0732) 51174
TELEX: 95142

Sintrom Electronics Ltd.*
Arkwright Road
Reading, Berkshire RG2 0LS
Tel: (0734) 85464
TELEX: 847395

Rapid Recall, Ltd.
6 Soho Mills Ind. Park
Woburn Green
Bucks, England
Tel: (6285) 270 72
TELEX. 849439

VENEZUELA
Componentes y Circuitos
Electronicos TTLCA C.A.
Aparlado 3223
Caracas 101
Tel 239-0820
TELEX 21795 TELETIPOS

*Field Application Location

INTEL CORPORATION, 3065 Bowers Avenue, Santa Clara, CA 95051 • (408) 987-8080
Printed in U.S.A./T-99/0979/15K DLR
AFN-00879A